Computers, Environment and Urban Systems 67 (2018) 29–40

Contents lists available at ScienceDirect

Computers, Environment and Urban Systems

journal homepage: www.elsevier.com/locate/ceus

Scalable evacuation routing in a dynamic environment☆

Kaveh Shahabi [a,b,*], John P. Wilson [c]

[a] Computer Science Department, University of Southern California, Los Angeles, CA, USA
[b] Google Inc., Mountain View, CA, USA
[c] Spatial Sciences Institute, University of Southern California, Los Angeles, CA, USA

ARTICLE INFO

Article history:
Received 8 August 2016
Received in revised form 19 August 2017
Accepted 26 August 2017
Available online xxxx

Keywords:
GIS
Evacuation
Traffic
Routing
Dynamic
Environment

ABSTRACT

In emergency management, tools are needed so we can take the appropriate action at different stages of an evacuation. Recent wildfires in California showed how quickly a natural disaster can affect a large geographical area. Natural disasters can create unpredicted traffic congestion or can temporarily block urban or rural roads. Evacuating a large area in an emergency situation is not possible without prior knowledge of the road network and the ability to generate an efficient evacuation plan. An ideal evacuation routing algorithm should be able to generate realistic and efficient routes for each evacuee from the source to the closest shelter. It should also be able to quickly update routes as the road network changes during the evacuation. For example, if a main road is blocked during a flood, the evacuation routing algorithm should update the plan based on this change in the road network. In this article major works in evacuation routing have been studied and a new algorithm is developed that is faster and can generate better evacuation routes. Additionally, it can quickly adjust the routes if changes in the road network are detected. The new algorithm's performance and running time are reported.

1. Introduction

How to plan a large-scale evacuation is a serious and an important research question. However, this question is not entirely answered. Currently we have the required computational power and good quality geospatial data to produce evacuation routes for large geographical areas. Furthermore, the advent of mobile platforms and smart cars have made it possible to reach out to individuals in seconds with life-saving news as might happen during a disaster. The missing element is the ability to generate efficient evacuation routes at large scale. Such a system should be able to monitor the evacuation and update the routes in case some of the initial conditions have changed.

Imagine there is a wildfire in Southern California. Fig. 1 shows a map of Southern California with major roads and historical wildfire incidents. The responsible agency generates evacuation routes to convey residents from each threatened neighborhood to specific shelters. These routes can be communicated with each neighborhood via a mobile or car app. Assuming none of the initial conditions change, everyone will get to a shelter in a reasonable and predictable time. The problem of generating these evacuation routes while minimizing the traffic congestion is what we call static evacuation routing. As the evacuation is happening, a number of things can change in the environment: (1) there could be background traffic that were not initially considered; (2) automobile accidents can temporarily affect the traffic flow; (3) changes to the road network such as a damaged bridge, a partially blocked road, or a flooded underpass can affect the road network; and (4) there could be more cars leaving one or more neighborhoods than we anticipated. Each of these situations can affect the evacuees. The public safety agency can learn about these changes either via the same mobile app (crowd sourced) or with the help of onsite public safety personnel. Updated evacuation routes could then be generated and pushed to affected motorists. The problem of generating and maintaining evacuation routes in a dynamic environment is called dynamic evacuation routing.

In this article, we propose a new solution to the dynamic evacuation routing problem. We focus on solutions that can be scaled to large geographical areas at least the size of a city. The remainder of this section provides an overview of the broader evacuation planning process with an emphasis on evacuation routing challenges. We then lay out the contribution and scope of our solution with respect to the evacuation planning problem. Section 2 reviews related work on evacuation routing. Section 3 formally defines the problem and explains the solution. Section 4 presents the experimental results. Section 5 offers conclusions.

1.1. Background

The evacuation problem can be decomposed into four phases: (1) mitigation; (2) preparedness; (3) response; and (4) recovery (Cova, 1999).

The **mitigation and preparedness** phases refer to the time before the incident actually happens. Since the nature of the disaster is not known at

☆ This study was completed as part of his graduate work before he joins Google.

* Corresponding author at: Computer Science Department, University of Southern California, Los Angeles, CA, USA.
E-mail address: shahabi@google.com (K. Shahabi).

http://dx.doi.org/10.1016/j.compenvurbsys.2017.08.011
0198-9715/

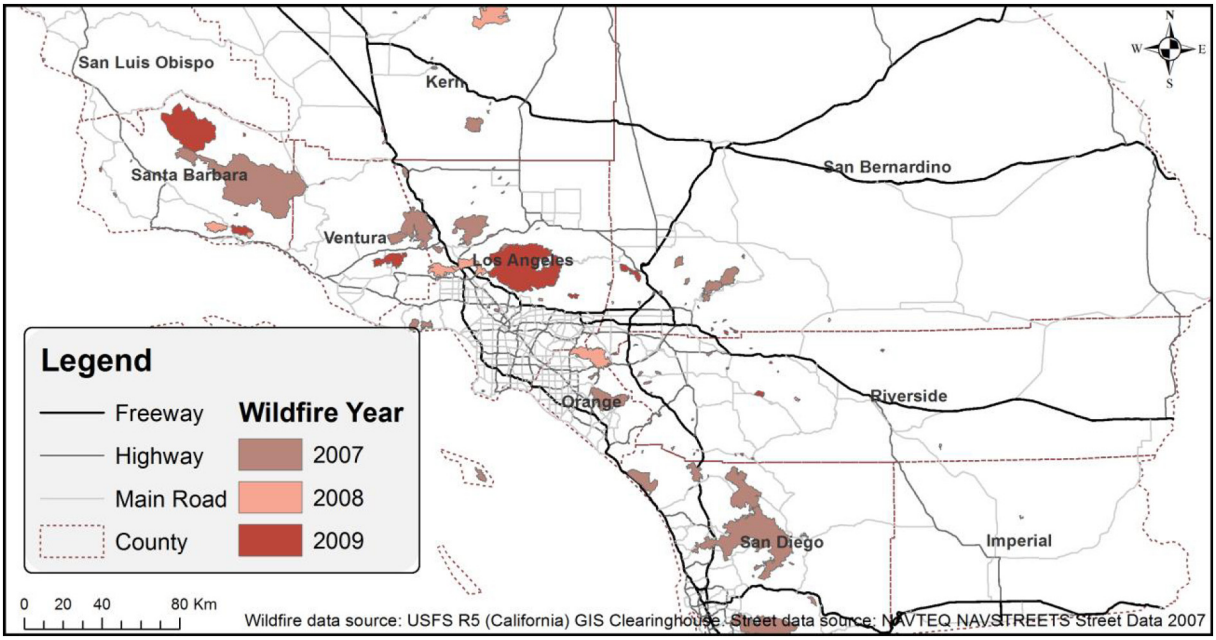

Fig. 1. Map of historical wildfires in southern California with major roads.

this stage, solutions to this sub-problem are less about the execution details and more focused on the surrounding geography. For example, Church and Cova (2000) presented a model to detect overpopulated neighborhoods that might face traffic congestion during an evacuation.

The **post-disaster recovery** is the stage where emergency personnel care for the affected population, maintain shelters, and assess the damage. For example, Yi and Ozdamar (2007) proposed a solution for evacuation response and support coordination during an emergency situation. The proposed model considers the food distribution to shelters and transportation of wounded people to medical centers as a commodity distribution problem. They formulated and solved the problem with mixed-integer LP.

The **evacuation response** refers to what needs to be done from the moment we learn about a disaster up until the time at which all of the potentially affected population is secured in safe areas. At this stage the location and time of the disaster and the evacuees are the key information.

Disasters can be static or dynamic. Dynamic disasters are those with changing behavior, location, or severity. Disasters like tsunami or terrorist attacks are considered static because the affected area, once known, is static. Other disasters like hurricanes, wildfires, and floods are dynamic since they move from one place to another. Of course given the circumstances, any disaster can become dynamic.

The **evacuees** are those residents who leave voluntarily whereas others may choose to shelter in place. The locations of the evacuees may be gathered from a variety of sources. In the U.S., high resolution population data is available through the U.S. Census Bureau (e.g. U.S. Census Bureau, 2010). However, these sources refer to residential populations and would not reflect the population distribution during the day. The LandScan team at Oak Ridge National Laboratory presented a method to extract high-resolution population data from a combination of geospatial datasets including satellite imagery (Bhaduri, Bright, Coleman, & Dobson, 2002). Kobayashi, Medina, and Cova (2011) presented a dynamic population model to improve and visualize diurnal population density based on public transport data.

1.2. Contribution

In previous works the static problem has been studied and several algorithms exist that can efficiently solve the problem (e.g. Lu, George, & Shekhar, 2005; Shahabi & Wilson, 2014). In this article we propose a routing solution for the dynamic environment. We formally define the problem and then develop a framework to solve both the static and dynamic versions of the problem.

Our main contribution is an iterative algorithm that can solve the evacuation problem in both static and dynamic environments. Our solution generates evacuation plans with better evacuation egress times compared to previous works. The secondary contribution is the ability to use a previously calculated evacuation plan to quickly find a new plan when a network change is detected.

2. Related work

The majority of the evacuation routing works in the literature can be divided into descriptive and prescriptive methods. Descriptive methods are those that visually simulate a given emergency situation. For example agent-based modeling (traffic simulation) visualizes what will happen during an emergency (Santos & Aguirre, 2004). Pel, Bliemer, and Hoogendoorn (2011) provided a thorough review of traffic simulation models for evacuation and looked specifically at the underlying formulation and psycho-behavioral assumptions of both commercial and academic traffic simulations. They categorized the models based on three choices: (1) evacuation participation and departure time; (2) destination choice; and (3) route choice. Prescriptive methods, on the other hand, determine the evacuation routing strategies to achieve some evacuation goal without necessarily performing a fine-scale simulation (Chiu, Zheng, Villalobos, & Gautam, 2007). In this section we briefly discuss major prescriptive evacuation routing methods.

Kwon and Pitt (2005) used the Dynasmart-P software (Mahmassani, Sbayti, & Zhou, 2004) to simulate the vehicular traffic for an evening event in downtown Minneapolis. They blocked different freeway ramps and observed the evacuation egress times. In one configuration they also implemented contraflow on freeways inside the study area. They concluded that it is feasible to study evacuation strategies using a dynamic network assignment model in a downtown-sized environment.

Xie, Lin, and Travis Waller (2010) defined and solved a new dynamic evacuation network optimization problem. In their work they considered both lane-reversal (contraflow) and crossing elimination jointly together and optimized the system for total evacuation egress time. The optimization was also conducted separately for network clearance

time. At a high level, the problem was formulated as a dynamic traffic assignment model and then solved with integrated Lagrangian relaxation and tabu search subject to the lane-reversal and crossing-elimination constraints. The experiments were performed for an area of 10-mile radius centered at the Monticello nuclear plant located in Minnesota.

Stepanov and Smith (2009) proposed a three-step LP-based urban evacuation routing algorithm that considered non-linear traffic flow. First, they generated k^{th} shortest paths from every evacuee point to every safe area. They next evaluated the quality of every path using a selected traffic model. And finally, they formulated the evacuation problem as an integer linear program (ILP). The ILP formulation considers the non-linear effect of every evacuee's path on others and then computes optimum evacuation routes. The same group previously studied both linear and non-linear analytical traffic models (Smith, 1991; Smith & Cruz, 2005) and has used them in their urban evacuation algorithm. Due to the computational complexity of the algorithm, the experiments were performed on a graph with just 11 nodes.

Bayram, Tansel, and Yaman (2015) recently published an evacuation planning solution that formulates the problem as a linear program that can be solved on a small graph. They used the U.S. Bureau of Public Roads (BPR) traffic model to estimate traffic times (U.S. Federal Highway Administration, 2014). They argued that evacuees may not willingly take a longer path to safety when there is a nearby shelter. However, they may take a slightly longer path. They formulated this assumption into their model and reported their experimental results. Nassir, Hickman, Zheng, and Chiu (2014) also recently published work that combines traffic signal optimization with evacuation routing. They initially decoupled the two problems without losing optimality and then formulated the routing problem as LP and solved it. Both of these fairly recent LP-based evacuation routing solutions can be used on graph sizes of up to 1000 nodes.

The Capacity Constrained Route Planner (CCRP) algorithm is a realistic evacuation routing solution that considers road capacity and population density (Kim, George, & Shekhar, 2007; Lu et al., 2005; Zhou et al., 2010). It also performs routing and scheduling simultaneously in order to improve the evacuation egress time. CCRP was the first algorithm that went beyond an LP or a simple shortest path solution. In essence, CCRP routes all evacuees to safety one at a time while constraining the road network (graph) to its road and intersection capacities. It can detect network bottlenecks (cuts) and can schedule evacuees in order to avoid graph saturation. The authors have continuously improved this algorithm to achieve better results. In recent work, they have developed a dartboard network cut-based approach to further improve the running time (Yang, Gunturi, & Shekhar, 2012). The main concern with CCRP is that it does not consider traffic congestion realistically. Therefore, its final evacuation routes are not optimized based on road network congestion. CCRP is also not suitable for evacuation in dynamic environments as it would need to be executed from scratch every time.

In another work, Pourrahmani, Delavar, Pahlavani, and Mostafavi (2015) transported evacuees to long-term shelters in multiple stages using public transportation. At each time interval, an OD matrix is generated from all interesting points, and a simulated annealing (SA) heuristic is used to solve the vehicle routing problem (VRP) at each stage.

We have briefly explained past work in the field of prescriptive evacuation routing. Some utilized LP-based solutions and others implemented heuristic algorithms for evacuation routing. While most methods consider traffic as a dynamic element, none has allowed the road network itself to vary through time. In the following section we formally introduce the evacuation routing problem in a dynamic environment and then propose a heuristic algorithm to solve it.

3. Method

In this section, we first formally define the problem. Next, we briefly explain how the Capacity-Aware Shortest Path Evacuation Router (CASPER) can be used to solve the static problem (Shahabi & Wilson, 2014). We then introduce two extensions to CASPER that would make it both faster and modular. With the help of these two extensions a new algorithm is designed called Dynamic CASPER (DCASPER). The following are the fundamental assumptions we have made in the DCASPER design:

1. We assume everyone starts an evacuation at the same time. In other words, we do not schedule the evacuation for some evacuees. We do, however, allow delays between cars coming from the same evacuation point. This technique is called metering and will dictate the automobile density on roads.
2. We assume that we have access to a traffic model in the form of a mathematical function that can predict the congestion on a street segment given total automobile density. Different traffic models can be used with DCASPER.
3. We assume all the cars at one evacuee point are inseparable and need to travel the same path. They generally remain close to each other throughout the evacuation.
4. The changes to the road network are not initially known; hence, the algorithm can only adjust once it learns about them. The only thing that cannot change during the evacuation are the shelter locations.

3.1. Problem statement

We model the dynamic evacuation routing problem as a path finding problem on a directed graph $G(E,V)$. We assume the graph elements are accessible in constant time. In addition, without loss of generality, we assume there is only one destination node (shelter) in the graph. The case with multiple destination nodes can be reduced to the single-destination case by introducing an artificial super sink node and connecting it to all the actual destination nodes with zero-cost edges.

The problem has five inputs: (1) the road network (graph); (2) the table of temporal road network changes; (3) a traffic model to predict congestion; (4) the evacuee locations; and (5) the destination point. The output is evacuation routes and predicted traversal times for each evacuee. The transportation network is represented as a graph $G(E,V)$ with $|V|$ vertices and $|E|$ directional edges ($E \subset V^2$). The changes in the network are modeled with the table of temporal changes. For example if one was modeling a wildfire evacuation, each time the fire blocks a group of roads, a new time interval would be created with the corresponding graph changes. Of course, the problem is more complex if there are frequent changes in the network.

Each graph edge e has positive initial impedance (imp) and capacity (cap). Since the road network can change during the evacuation, neither of these values are constant. For example, the impedance could be the initial road segment traversal time (without traffic) and the capacity could be the number of lanes. This edge e may no longer be accessible 20 min after the evacuation starts due to fire hazard. We model this road inaccessibility as a decrease in the number of lanes (capacity), which means the capacity of edge e becomes zero after minute 20. Generally speaking, we are allowing each graph edge to change its values at some time after the start of the evacuation. In other words, at each time interval, a subset of edges can change their impedance and capacity values to larger or smaller values.

$$\begin{aligned} &GC = \{gc = (time, e, imp, cap) | time \geq 0, e \in E\} : \text{set of all graph changes} \\ &imp_{gc}(e), cap_{gc}(e) : \text{impedance and capacity of edge e after graph change } gc \\ &\forall e \in E, gc \in GC \; imp_{gc}(e) \geq 0, cap_{gc}(e) \geq 0 \end{aligned} \quad (1)$$

The set S holds all the evacuee points and their populations. Each evacuee point s has a positive weight $w(s)$ (Eq. (1)). For example, an evacuee point could be a census block group or a residential building. The weight is the number of vehicles at that point. Each evacuee point can have many vehicles.

$$\forall s \in S, S \subset V \; w(s) > 0 \quad (2)$$

Each evacuee point s generates a different density on each edge. If we assume that one car leaves the neighborhood every 2 s. Edge e is 20 s in length, and evacuee s has 300 cars, we can calculate the density based on the initial delay between each vehicle. So we have $delay = 2\ s$, $imp(e) = 20\ s$, $w(s) = 300$. Eq. (2) computes the density from an evacuee point on an edge as the number of cars that fits on that edge: $den(s,e) = 20/2 = 10$. A path P_s is an ordered set of edges. It starts at s and ends at a shelter (t). Therefore, the total density on edge e is the sum of all densities from all paths that pass through e (Eq. (3)).

$$den_{gc}(s,e) = \min\left(\frac{imp_{gc}(e)}{delay}, w(s)\right) \tag{3}$$

$$den_{gc}(e) = \sum_{\forall s \in S, P_s \in \wp, P_s \subset E,\ s,e \in P_s} den_{gc}(s,e) \tag{4}$$

The traffic model is a function with two parameters (d,c) (Eq. (4)). This function estimates the congestion based on edge total density (d) and capacity (c). The traffic model outputs a number between 0 and 1: $\mathcal{T} = 1$ means there is no traffic congestion and $\mathcal{T} = 0$ means infinite congestion. Eq. (5) calculates the cost of traversing an edge. From there, we calculate the cost of each path. The first term in the path cost formula (Eq. (6)) accounts for the extra time imposed by the initial vehicle delays and the second term sums the cost of each edge of the path. The summation in Eq. (6) needs to know at what time interval each evacuee arrives at each edge to be able to sum up the correct edge costs. We call this the arrival time of evacuee s to edge e over its path: $arrival(s,e,P_s) \in GC$. The objective is to minimize the final evacuation egress time (Eq. (7)).

$$\forall d, c \in \mathbb{R}^+\ \mathcal{T}(d,c) \in (0,1], \mathcal{T}(0,c) = 1, \frac{\partial}{\partial d}\mathcal{T}(d,c) \le 0, \frac{\partial}{\partial c}\mathcal{T}(d,c) \ge 0 \tag{5}$$

$$cost_{\mathcal{T},gc}(e) = \frac{imp_{gc}(e)}{\mathcal{T}\left(den_{gc}(e), cap_{gc}(e)\right)} \tag{6}$$

$$cost_{\mathcal{T}}(P_s) = delay \times w(s) + \sum_{e \in P_s, gc = arrival(s,e,P_s)} cost_{\mathcal{T},gc}(e) \tag{7}$$

$$EvcTime = \max\{\ cost_{\mathcal{T}}(P_s) \mid P_s \in \wp\ \},\ \textbf{Objective}: minimize\ EvcTime \tag{8}$$

When evacuation routes share an edge, they are going to affect each other's traffic. The density on the shared edges will increase which results in higher egress times. In order to lower the egress time, the routing algorithm needs to find the shortest paths to safety while minimizing the traffic congestion. Table 1 describes the remainder of the symbols which are used in this article.

3.2. Traffic model

We have thus far discussed how we use traffic models without describing a model. Traffic congestion and modeling has been studied by traffic researchers both in theoretical and experimental form. Therefore, there are many approaches and models to choose from (e.g. Greenberg, 1959; Helbing, 2001; Holden & Risebro, 1995; Malone, Miller, & Neill, 2001; Smith, 1991; Smith & Cruz, 2005). For the purpose of this research, we used an empirical model called the Power Traffic Model. It can be formulated as follows:

$$\begin{aligned} &Power: \mathcal{T}(d,c) = 1 - \gamma\sqrt{d} \times e^{-\varepsilon c} \\ &where\ \gamma\ and\ \varepsilon\ are\ constants \\ &\gamma = 0.02261, \varepsilon = 0.01127 \end{aligned} \tag{9}$$

The Power model has a flexible formulation and is the result of empirical curve fitting (Shahabi & Wilson, 2014). We defined the constants such that a single-lane, one-way road segment with a car density of 500 would have $\mathcal{T} = 0.5$ and a three-lane segment would have $\mathcal{T} = 0.51$. Both constants, γ and ε, will remain unchanged even if the road network changes. This model improves the traffic congestion predictions and the evacuation egress time estimates. Experimenting with different traffic models is beyond the scope of this work. However, interested readers are encouraged to look at the extensive review of traffic flow models provided by Leutzbach (1988).

3.3. Static solution

Our solution to the dynamic evacuation problem is built on top of the static solution described in Fig. 2. This schematic diagram illustrates the CASPER architecture and provides a high-level description of the process.

Initially, all inputs are loaded: evacuees, shelters, and the traffic model. Then a separate module called the Capacity-Aware Reverse Map Analyzer (CARMA) is utilized to build a graph from the GIS data. CARMA is also responsible for building a set of heuristics on the graph that improves the overall running time. To generate the evacuation routes, a routing algorithm similar to A* (Hart, Nilsson, & Raphael, 1968) is implemented. CASPER finds the shortest route for each evacuee while considering the predicted traffic times and automobile density. Once the route is found, CASPER reserves the route for the corresponding evacuee and then moves on to the next evacuee. Every time a route is reserved, the predicted traffic times are changed and consequently the graph heuristics are invalidated. Because the routing process heavily relies on these heuristics, CASPER gets slower and slower as it reserves more routes. Therefore, it is necessary to refresh these heuristics. CARMA has a mechanism to detect when a good time to rebuild the heuristics occurs. After rebuilding the heuristics, CASPER resumes to process more evacuees. This back-and-forth process between CASPER and CARMA continues until all evacuees are processed. Algorithm 1 outlines the CASPER evacuation routing system that solves the static evacuation problem.

Algorithm 1. Static Evacuation Routing

```
Input: G, S, t, 𝒯
𝒫 ← ϕ
∀s ∈ S, w(s)>0 do {
  1. route P_s ← FindEvacuationRoute_𝒯(G, s, t)
  2. 𝒫 ← 𝒫 ∪ {P_s}
  3. Reserve P_s for s. ∀e ∈ P_s : den(e) is increased by den(s,e)
  4. Update edges cost: ∀e ∈ P_s recalculate cost_𝒯(e)
  5. Collect invalidated heuristic values
  6. If invalidated heuristic ratio > 20% {
     6.1. Run CARMA_𝒯(G, S, t)
  7. }
}
Output: 𝒫
```

CARMA internally builds a shortest path tree (SPT) to perform the required task. The *SPT* is built with a single Dijkstra over the reversed G with t as the starting vertex. Then CARMA updates the heuristic value, $h(v)$ for all of the vertices. The *distance*(*SPT*, s, t) function mentioned in Algorithm 2 (line 2) is the path cost of s to t on the tree. Lastly, CARMA sorts all evacuees based on their estimated shortest path distance from the shelter in reverse order. This way, the furthest evacuee will be processed next by CASPER. Algorithm 2 outlines how CARMA works.

Table 1
Descriptions of symbols.

Symbol	Description
$DV \subset V$	Dirty vertices set
$DE \subset E$	Edges with changed costs (dirty edges)
$g(v) \in \mathbb{R}^+$	Cost from source to vertex v
$h(v) \in \mathbb{R}^+$	h value of vertex v that serves as a lower bound of the cost to destination
SPT	Shortest path tree; also an algorithm that finds such a tree
S'	The set of unprocessed evacuee points

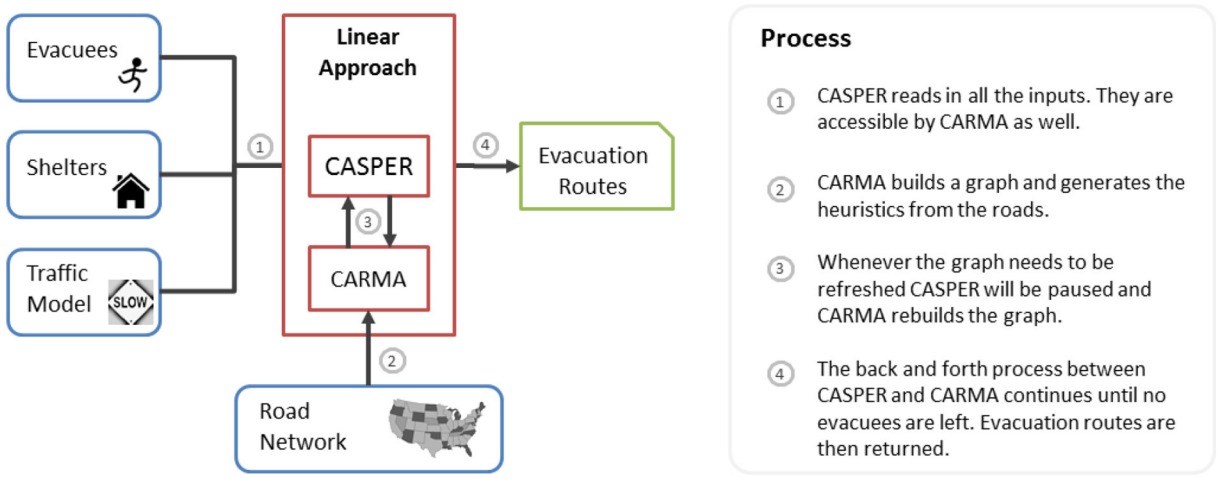

Fig. 2. Static Evacuation Router architecture. This diagram demonstrates the major steps of the routing process.

Algorithm 2. CARMA (short version)

Input: G, S, t, $\mathcal{T}$
1. Build the shortest path tree: SPT ← BuildSPT$_{\mathcal{T}}$(G, t)
2. ∀v ∈ V: h(v) ← distance$_{\mathcal{T}}$(SPT, v, t)
3. Sort S (evacuees) descending based on distance to shelter: distance$_{\mathcal{T}}$(SPT, s, t), s ∈ S

Output: heuristics

There are two advantages in running CARMA multiple times during the evacuation routing process: (1) it rebuilds the graph heuristics which improves the overall running time; and (2) it re-orders evacuees based on their distance to shelters. Conceptually speaking, rebuilding the heuristic values is like pre-paying for the routing costs in one batch. This is why running CARMA statistically helps with the overall running time. Sorting evacuees often helps with overall evacuation egress time. One of the key challenges with evacuation routing is the combinatorial effect of these routes on each other. Every time a route is reserved it affects the travel time of other routes because it changes the car density on the graph. This combinatorial effect makes it very difficult to solve the problem optimally.

3.4. Dynamic shortest path tree

As discussed earlier, CARMA is responsible for building and maintaining an SPT. CASPER uses these heuristic values, $h(v)$, to find evacuation routes faster. The standard way to regenerate these values is to build the SPT from scratch but this is going to be slow. In this section we discuss an alternative method to refresh the outdated heuristics.

The majority of the computational work in CARMA is building the SPT. This step is relatively time consuming. It is also repetitive as not all parts of the graph are always affected. Usually about 20% of the edges are affected after a few route reservations. Therefore, the new SPT is going to mostly look like the old one. This problem is called the dynamic shortest path tree (DSPT) in the computer science literature. The algorithms that update the tree in the case of an increased cost are called semi-dynamic and algorithms that can update the tree whether the cost is increased or decreased are called fully-dynamic.

Ramalingam and Reps (1996) proposed a fully dynamic SPT incremental algorithm that can handle edge weight updates. For each graph vertex they maintain a list of distances-to-source based on all outgoing edges. After every edge weight update, the distances are updated incrementally until the SP tree is re-constructed. Frigioni et al. (Frigioni, Ioffreda, Nanni, & Pasquale, 1998; Frigioni, Marchetti-Spaccamela, & Nanni, 1998; Frigioni, Marchetti-Spaccamela, & Nanni, 2000) presented fully dynamic and semi-dynamic SPT algorithms along with their theoretical proofs. Their recent work, FMN, shows that SP trees can be maintained in $O(logn)$ time for a group of special graphs and in $O(\sqrt{m}\, logn)$ time for general graphs. The reported complexities are per output update. They have also showed that if insertions or deletions of edges are allowed, then similar amortized bounds hold. In reported experimental results, it has been shown that while FMN has a lower time complexity, it is outperformed in most cases by Ramalingam and Reps's algorithm. Chan and Yang (2009) presented an in-depth review of fully- and semi-dynamic SPT algorithms and compared their running times. They also improved an existing dynamic algorithm, BallString (Narváez, Siu, & Tzeng, 2001), and showed that the corrected version (MBallString) outperforms other DSPT algorithms.

We used the MBallString algorithm with some minor modifications because it provided a better fit with our GIS setup. CARMA initially gets all the changed edges (*DE*) with their new costs. Then using the old tree, it finds all the locally affected vertices of these edges. The locally affected vertices (*DV*) are vertices with shortest paths that have likely changed. In other words, the path from a locally affected vertex to the tree root contains at least one dirty edge. The locally affected vertices are referred to as *dirty* and other vertices as *clean* here. All dirty vertices are removed from the tree. Next we look for dirty vertices that have at least one clean adjacent vertex. These are called boundary vertices (*BV*). We enqueue all boundary vertices to a priority queue. Note that leaf vertices are clean otherwise they must have been removed from the tree. The remainder is very similar to the original SPT algorithm. We dequeue a vertex, process its adjacent vertices, and continue until the missing parts of the SPT are rediscovered. We generally do not need to enqueue clean vertices unless we find a lower cost for them than the one in the original SPT. Algorithm 3 outlines the fully-dynamic SPT implementation.

Algorithm 3. Fully-dynamic SPT

Input: G, SPT, $\mathcal{T}$
1. DE ← ExtractDirtyEdges(G), DE ⊂ E
2. DV ← FindLocallyAffected(SPT, DE), DV ⊂ V
3. SPT' ← Clone(SPT)
4. SPT' ← RemoveVertices(SPT', DV)
5. BV ← FindBoundaryVertices(DV, SPT'), BV ⊂ DV
6. PQ: create priority queue
   For each boundary vertex, we find the shortest distance to root from adjacent clean vertex.
   This shortest distance is used as the vertex priority.
7. ∀v ∈ BV: Enqueue(PQ, v, distance$_{\mathcal{T}}$(SPT, v, t))
8. While PQ ≠ ϕ {
   8.1. (v, dist) ← Dequeue(PQ)
   8.2. SPT' ← SPT' + {v}
   8.3. ∀u ∈ Adjacent(reverse of G, v), e = (u, v) ∈ E {
      8.3.1. newdist ← dist + cost$_{\mathcal{T},gc}$(e)
         First we check if a fully-dynamic tree reconstruction is needed. If a better cost for an existing vertex is found then we have to enqueue it.
      8.3.2. **If u ∈ SPT' AND newdist < distance$_{\mathcal{T}}$(SPT, u, t) then**
         8.3.2.1. SPT' ← SPT' - {u}
         8.3.2.2. Enqueue(PQ, u, newdist)
      8.3.3. If u ∉ SPT' then EnqueueOrUpdate(PQ, u, newdist)
   8.4. }
9. }

Output: SPT'

If there is at least one edge where the cost is decreased, then the fully-dynamic SPT algorithm is utilized; otherwise, the semi-dynamic version is used. The two DSPT algorithms only differ in vertex rediscovery (Algorithm 3, line 8.3.2). In our experiments with different DSPT algorithms, this hybrid implementation outperformed static SPT in all cases. DSPT will be as slow as static SPT once we have more dirty vertices than clean ones.

3.5. Iterative routing

In the previous section, we have shown how a DSPT algorithm can improve CASPER running time. In this section we focus on improving final evacuation egress time. As discussed earlier, the evacuation routing problem is fundamentally a difficult problem because of the combinatorial effect of evacuation routes on one another.

While it is possible to improve CASPER's evacuation egress time with linear or dynamic programing, such designs will not be scalable and hence not useful for a realistic scenario. At the same time the current greedy approach is not providing us with any other option to marginally improve the final egress time. Therefore, we decided to study iterative algorithms. Minton, Johnston, Philips, and Laird (1992) proposed a heuristic solution to the satisfaction problem that is faster than a simple backtracking algorithm. It starts with a random assignment for all variables in the satisfaction problem. Then a repair method is iteratively applied until an acceptable solution is found. The repair method follows a heuristic: at each repair step it minimizes the conflict among individual variables. They showed that this conflict-minimizing heuristic is significantly better than previous approaches, including backtracking. The evacuation routing problem is not a satisfaction problem and the idea of a conflict is not clear but these are not fundamental differences.

In another work, Li, Zhu, Li, Wu, and Zhang (2015) proposed a novel building evacuation algorithm based on game theory and Monte Carlo optimization. They modeled the building evacuation as an n-person non-cooperative game. First, the distance to each exit is calculated for all rooms. Then, the payoff costs for individuals are calculated based on corridor congestion. Next, they iteratively assign evacuees to exit routes until the system reaches the Nash equilibrium. This algorithm requires many calculations on multiple paths and hence, it is not immediately scalable to our urban evacuation problem. We hypothesized that the idea of competing evacuees in a building evacuation is similar to conflicting variables in a satisfaction problem and can be utilized for an urban evacuation problem.

Before we lay out the iterative design, we need a few more definitions. First, we need a definition for two conflicting or competing paths. Two paths that share many edges indicates that they compete over the same region of the graph. This kind of competition adds to the complexity of the evacuation problem. Therefore, the conflict of two paths is defined according to how much they overlap with each other. The overlap is calculated based on the cost of shared edges between the two paths (Eq. (9)). Our experiments showed that a 40% overlap is a good indication that two paths are conflicting. We also learned from the experiments that certain paths do not have the potential to be improved. We identify these paths by comparing the final path cost, the cost immediately after reservation, and the cost predicted by the SPT. If any of these differences for a path is >15% of the final egress time, then this path can likely be improved. Eqs. (9) and (10) show how we calculate the *disadvantage* and *conflict* ratios.

$$Conflict_{\mathcal{T}}(P_1, P_2) = \sum_{e \in P_1 \cap P_2} \left(cost_{\mathcal{T}, gc}(e)\right) \Big/ \sum_{e \in P_1} \left(cost_{\mathcal{T}, gc}(e)\right), gc : cte \tag{10}$$

$$Disadvantage_{\mathcal{T}}(EvcTime, P_s) = \frac{max(ReserveCost_{\mathcal{T}}(P_s) - Distance_{\mathcal{T}}(SPT, s, t),\ \ Cost_{\mathcal{T}}(P_s) - ReserveCost_{\mathcal{T}}(P_s))}{EvcTime} \tag{11}$$

Algorithm 4 outlines the iterative approach for solving the evacuation routing problem. For the first round, the greedy CASPER algorithm finds evacuation routes for all evacuees. We then look for paths with a disadvantage and select the corresponding evacuees. On the next iteration, greedy CASPER finds paths for the selected evacuees and merges new results with the non-selected paths. The iteration continues until the egress time can no longer be decreased.

Algorithm 4. Iterative CASPER

Input: G, S, t, $\mathcal{T}$
1. $S' \leftarrow$ Clone(S)
2. $\mathcal{P} \leftarrow \phi$
3. Global variables: EvcTime $\leftarrow \infty$, MaxSize $\leftarrow |S|$
4. Constants: ItrRatio $\leftarrow 0.6$, DisadvantageRatio $\leftarrow 0.15$, ConflictRatio $\leftarrow 0.4$
   After initializing some variables, we start the iteration loop. At each step we call the greedy CASPER with the selected evacuees.
5. While $S' \neq \phi$ {
   5.1. New$\mathcal{P} \leftarrow$ GreedyCASPER$_{\mathcal{T}}$(G, S', t)
   5.2. PrevEvcTime $\leftarrow$ EvcTime
   5.3. EvcTime $\leftarrow$ max { $cost_{\mathcal{T}}(P_S) \mid P_S \in \mathcal{P} \cup$ New$\mathcal{P}$ }
   5.4. If EvcTime $\leq$ PrevEvcTime then $\mathcal{P} \leftarrow \mathcal{P} \cup$ New$\mathcal{P}$ **ELSE** terminate iteration
   5.5. MaxSize $\leftarrow$ MaxSize * ItrRatio
   5.6. $S' \leftarrow \phi$
   At this step we select the evacuees that can potentially be improved
   5.7. $\forall s \in S, \exists P_S \in \mathcal{P}$ {
      5.7.1. If Disadvantage$_{\mathcal{T}}$(EvcTime, P_S) $\geq$ DisadvantageRatio AND $|S'| \leq$ MaxSize {
         If the difference between predicted cost, reserved cost, and final cost for a path is larger than 15% of EvcTime then it's selected for iteration. Paths that overlap with it by more than 40% are selected too.
         5.7.1.1. $S' \leftarrow S' \cup \{s\} \cup \{ s_2 \neq s \mid Conflict_{\mathcal{T}}(P_S, P_{S_2}) \geq ConflictRatio \}$
      5.7.2. }
   5.8. }
   5.9. Sort S' by $cost_{\mathcal{T}}(P_{S_2})$
   The evacuee with the longest path will be processed first in the next iteration
6. }

Output: $\mathcal{P}$

At line 5.5 an upper bound is set for the number of selected evacuees. This bound is exponentially decreased at every step by a factor <1. This guarantees that the algorithm terminates in at most $O(|S|)$ operations, the same as the greedy CASPER.

3.6. Dynamic evacuation routing

In the previous sections, a new algorithm, called DCASPER, was outlined which is flexible to a changing environment. The new DSPT implementation enables CARMA to quickly update its internal data structure after any kind of road network change. The iterative design made the entire system a modular algorithm. You can feed in a set of evacuation routes and have the iterative CASPER improve the routes. The same iterative algorithm can also be used to improve routes after a change in the road network.

Fig. 3 demonstrates how DCASPER solves the dynamic evacuation routing problem with a modular design. The beginning steps of DCASPER are similar to the static solution. DCASPER solves the problem like any static evacuation routing problem and outputs the routes. Next, it reads in the first set of network changes. For example, one network change could be that 20 min after the evacuation is started, a portion of a freeway is modified to allow a higher speed limit but some of its ramps are now closed. To model this, we recalculate the cost of edges associated with the freeway. We also set the edge cost of closed ramps to infinity. The new information is first passed to CARMA to update the graph. Next, we move all evacuees on their paths for 20 min. Lastly, we invalidate all paths that cross the closed ramps. The remaining paths are fed back to DCASPER to be reprocessed.

As DCASPER is processing each change, it also needs to track evacuee locations and their partial paths. It needs to know if an evacuee has already reached a shelter, is trapped in a disconnected sub-graph (stranded), or still needs to be routed at the next step. Lastly, it is necessary to calculate the final egress times based on correct edge costs and traffic densities at the appropriate interval. To be able to compare DCASPER with a baseline, we have created four different variations of the algorithm:

- **DisableDCASPER**: This version assumes that there are no network changes. It behaves like the static CASPER algorithm. The results will give us an understanding of how much the network dynamicity is affecting the evacuation egress time and the algorithm running time.
- **SimpleDCASPER**: Similar to DisableDCASPER but it assumes all network changes occur at the beginning and will continue to exist forever. By doing so, we are effectively giving SimpleDCASPER some additional knowledge about the future of the road network. Since this information is known at the beginning, SimpleDCASPER can avoid all of them at once. There is no need to iterate over all network intervals. The results from this version will help us understand the best possible scenario achievable with a static algorithm.
- **FullDCASPER**: This version respects all network changes and their start times. After every network change, it deletes the graph, heuristics, and paths and then runs iterative CASPER from scratch. Since this version does not rely on existing paths, it serves as our naïve approach.
- **SmartDCASPER**: This is the version we have discussed in this section. It can quickly update its internal data structure and then it reprocesses the affected paths. The main advantage of SmartDCASPER over FullDCASPER is that it only needs to focus on updating a small set of paths. Therefore, it can potentially produce better evacuation routes with less computation.

DCASPER is a complete sub-optimal algorithm. It can find a path for each evacuee while satisfying all network changes if such a path exists. However, it is not guaranteed that the final evacuation plan would have the lowest possible egress time.

3.7. Complexity analysis

We already know that the static greedy CASPER has a running time similar to the Dijkstra algorithm: $O(|S| \times (|E| + |V|\log(|V|)))$. Adding DSPT to CARMA does not change its computational complexity. In the worst case scenario, the entire tree still needs to be traversed, which is the same as running a single Dijkstra. Even though the iterative design can slow down CASPER, it does not change its computational complexity either. Since the *ItrRatio* is <1, at each iteration, not all evacuees are processed. More specifically, at each iteration we have:

$$\#of\ processed\ evacuees\ at\ iteration\ i = |S| \times ItrRatio^{i}, ItrRatio<1 \tag{12}$$

$$Total\ possible\ iterarions: |S| \times ItrRatio^{i_{max}} \le 1 \Rightarrow i_{max} \ge \frac{log\left(\frac{1}{|S|}\right)}{\log(ItrRatio)} = \log_{ItrRatio}\left(\frac{1}{|S|}\right) \tag{13}$$

$$\begin{aligned} Total\ processed\ paths &= \sum_{i=0}^{i_{max}} |S| \times ItrRatio^{i} \\ &= |S| \times \frac{ItrRatio^{i_{max}} - \frac{1}{ItrRatio}}{1-\frac{1}{ItrRatio}} = |S| \times \frac{\frac{1}{|S|} - \frac{1}{ItrRatio}}{1-\frac{1}{ItrRatio}} = \frac{|S| - ItrRatio}{1-ItrRatio} \\ &= O(|S|) \end{aligned} \tag{14}$$

The total number of processed paths varies linearly with the total number of evacuees (Eq. (13)). Hence, the iterative CASPER has the same complexity as the greedy static CASPER. To solve the dynamic evacuation problem, DCASPER needs to adjust routes every time there is a new dynamic change. We have a total of $|GC|$ changes in the road network and therefore can express DCASPER complexity as:

$$DCASPER\ complexity = O(|GC| \times |S| \times (|E| + |V|\ log(|V|))) \tag{15}$$

4. Results

The new algorithm was compared with static CASPER in terms of algorithm running times and evacuation egress times. The evacuation program is implemented with C++ as a single-threaded plug-in for ArcGIS Desktop® (Shahabi, 2015a). The source code is also publicly available (Shahabi, 2015b). Everything from loading data, storage, accessing the transportation network, and visualization is abstracted by the ArcGIS API® (Esri, 2015). The implementation supports simple turn restrictions, multiple destination points, different impedance metrics (time, length, etc.), multiple traffic models, timed dynamic network changes, and shelter capacity. All the experiments were performed on a dedicated 64-bit Microsoft Windows® 10 machine with Intel i5 CPU (1.9 GHz) and 8 GB of memory. The few seconds taken to load inputs, visualize, and store the results are excluded.

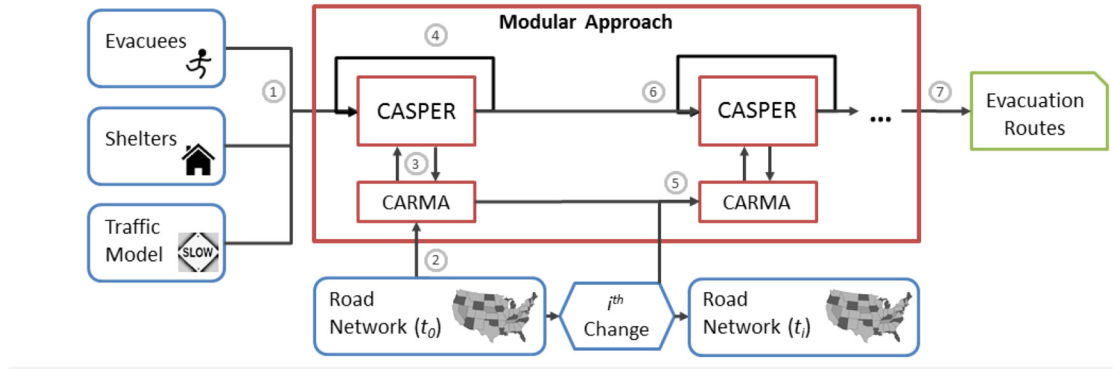

Fig. 3. DCASPER architecture. This diagram demonstrates how a modular design improved the original static algorithm in order to solve the dynamic evacuation routing problem.

4.1. SoCal fire setup

We used the southern California wildfire records from 2008 and 2009 to create eight different wildfire polygons that served as the evacuation zones. These polygons represented the extent of the fires. The schools reported in the US public schools layer are used as shelters for the experiments that follow (USC Spatial Sciences Institute, 2014). For each experiment, all schools that are outside the evacuation zone and at least 10 km away from it, are considered a shelter. For each wildfire polygon, we also created another small polygon to represent the dynamic changes to the road network. All roads inside a dynamic change polygon were set to get fully or partially blocked 10–30 min after the evacuation starts. To select the dynamic change areas, we picked main road segments that are useful for evacuation. However, we have avoided creating blockages on major roads as that would create major cuts in the network at which point evacuation would be near impossible.

Fig. 4 visualizes both types of polygons for each county along with public schools. We used the complete southern California road network to build the graph. The road network includes highways and local roads. The resulting graph had about 750,000 vertices and 1.8 million edges.

The evacuees were all of the people who lived inside each of the wildfire polygons. We used the 2010 Census Block Group dataset to get the locations and population counts for the evacuees. Commonly, wildfires occur in rural areas where population density is low, but this would not allow us to test the DCASPER scalability and performance. We added three other urban fire polygons in highly populated areas: the Los Angeles, Santa Ana, and San Diego downtown areas. Fig. 5 visualizes the evacuees and shelters for the Santa Ana scenario. Finally, we merged smaller areas to generate even larger evacuation scenarios. In total we generated 13 evacuation scenarios that cover a vast geographical area as well as different population densities, road network structures, and dynamicity. Table 2 lists these 13 scenarios.

Table 2 is split into two parts: urban and rural scenarios. Each part is sorted according to fire polygon size. For each scenario, the evacuating neighborhoods are what the routing algorithm receives as *S*. The 'vehicle count' gives us an idea of how much traffic this scenario would generate. For the rural areas we set the dynamic polygons to permanently block the intersecting roads. As an example, a dynamic polygon is blocking 19 km of road 10 min after the evacuation starts and continues until infinity for the Santa Barbara (SB) wildfire scenario. Depending on the setup, some of these scenarios produce unusually high evacuation egress times due to network saturation. Therefore we excluded them from measurements.

4.2. DSPT implementation

We next present the comparison between the two SPT algorithms. Aside from the SPT implementation, we can also change the invalidated heuristic ratio which affects the number of times CARMA is updating the tree in one scenario. The lower the ratio, the greater the number of tree updates. We can also choose to sort evacuees after each DSPT run or only once before the initial CARMA loop. In total, these options gave us six different setups (Table 3). The *Dspt* prefix in the setup name indicates the dynamic SPT implementation and the *CDspt* prefix indicates dynamic SPT implementation with continuous evacuee sorting. The *C0X* prefix specifies the invalidated heuristic ratio that was used. For the sake of this comparison, we ignored all dynamic polygons and completed just one iteration (greedy CASPER) to solve the static problem.

In this experiment, the average of the running and final egress times were calculated over 25 test cases. They consist of all fire scenarios each with two different initial car densities. The EvcTime improvement is calculated as the percentage of improvement one setup has over the one with the worst egress time in each scenario. We used the same calculation method for the speed up percentage. As can be seen from Table 3, CDsptCASPER has the best combined running time speed up. The CDsptCASPER_C02 variation has an average 20.3% speed up compared to the slowest setup. Since CDspt sorts evacuees every time, it ends up with a relatively smaller tree; hence, it is slightly faster than Dspt.

4.3. Iterative design

In order to test the iterative CASPER, we started with all 13 fire cases and three different initial car densities. We then created 10 different variations of CASPER based on the invalidated heuristic ratio, SPT implementation, evacuee sorting policy, and iterative design. The resulting 340 experiments have been measured similar to previous experiments. Table 4 summarizes the measured average improvements and speed ups. The *Itr* and *CItr* prefixes indicate the iterative design and the *CItr* prefix also specifies the continuous evacuee sort policy. Both iterative CASPER implementations have the faster DSPT implementation embedded in them.

There is a clear distinction between the iterative CASPER and the non-iterative CASPER in egress time. As predicted, iterative CASPER is slower by a constant factor. The measured average running time is about 20% slower than that of the non-iterative CASPER (Table 4, column 7). Both ItrCASPER and CItrCASPER performed well with different evacuation scenarios and created relatively better routes. On average, ItrCASPER improved the EvcTime by 8%. Another interesting observation was that continuously sorting evacuees makes CASPER, both iterative and non-iterative, run slightly faster. Overall, the choice comes down to algorithm speed verses route quality.

4.4. Dynamic environment

Thus far we have ignored the dynamic polygons in our experiments. We present the results for the dynamic evacuation routing problem here. As defined earlier, there are four different variations of DCASPER, 13 fire cases, and three different initial car densities. After removing a few very fast scenarios, we end up with 154 experiments. Table 5 presents the average performance and quality of the said experiments.

We have selected slightly higher initial car densities compared to the iterative experiment for two reasons: (1) to avoid running the same experiments twice; and (2) to generate scenarios with relatively lower traffic congestion. High traffic congestion forces DCASPER to always start from scratch after every network change. This will not allow us to study the performance of different DCASPER variations. For the same reason, we have also created small dynamic polygons because we wanted only a small portion of the graph to change. Another advantage of having small dynamic polygons is that we will have less stranded evacuees.

DisableDCASPER is the only setup that stranded no evacuees because it ignored the dynamic polygons. This is an entirely unrealistic dynamic solver that we made for comparative purposes (i.e. as ground truth). For the same reason DisableDCASPER is the fastest algorithm. SimpleDCASPER is also an unrealistic solver because it knows about all the dynamic polygons in advance. This gives it an unfair advantage over the other setups. Because of this extra knowledge, most evacuees will not get stranded as they follow their paths. Both DisableDCASPER and SimpleDCASPER have relatively low running times because they do not need to dynamically adjust the evacuation routes after each dynamic change.

SmartDCASPER and FullDCASPER are realistic solvers. They both consider all dynamic polygons and have to adjust their routes after the network changes are detected. The experiments do confirm that SmartDCASPER has indeed lower running time than FullDCASPER (Table 5, Column 6). These experiments also show that SmartDCASPER produced better evacuation plans (Table 5, Column 4). As evacuees are moving toward shelters, there are fewer unique routes left for them. Fewer unique routes means that the combinatorial characteristic of the problem is intensified. Therefore, the problem is harder to optimize. Since FullDCASPER solves the problem from scratch after every dynamic event, it may not find the same good routes it found in previous iterations.

In order to better compare different DCASPER methods, we can look at the affected neighborhoods. For example, Fig. 6 shows how one

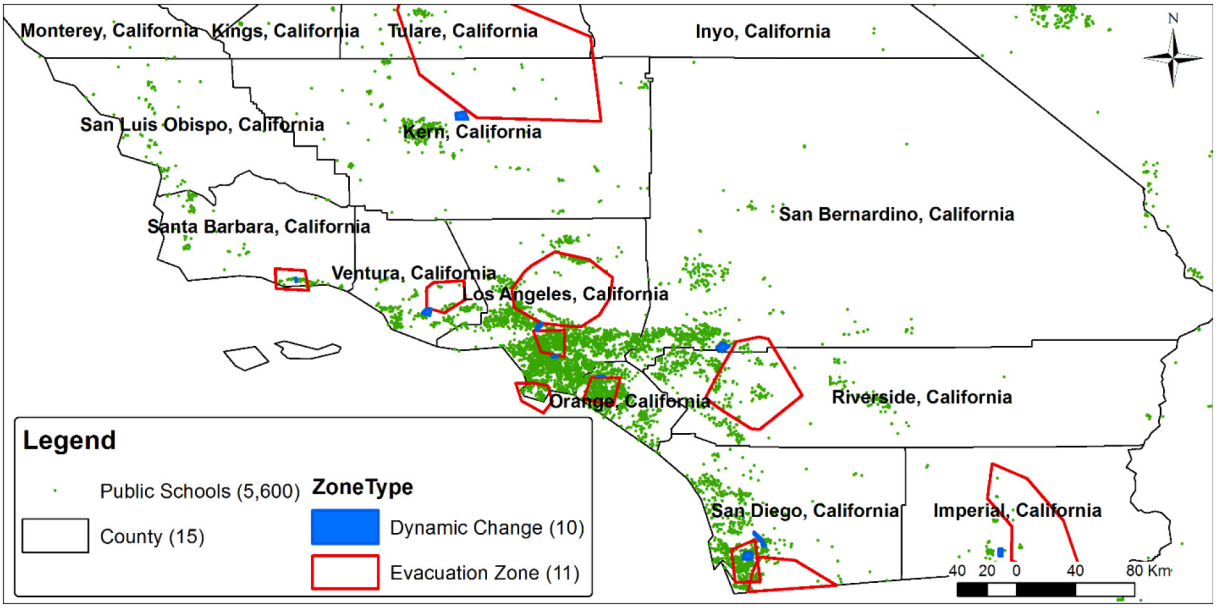

Fig. 4. Map of southern California with public school locations and realistic wildfire evacuation zones.

neighborhood in Santa Barbara, CA would be evacuated with different DCASPER methods. DisableDCASPER ignores road restrictions so it drives the neighborhood through the dynamic blockage zone. SimpleDCASPER knows about the blockage in advance and generates a route that goes around the blockage toward the same public school. SmartDCASPER, however, does not know about the blockage until it happens. Therefore, it has to correct the route midway through the evacuation. It initially goes toward the same public school. After the blockage, it makes a U-turn and moves toward another public school to the west. FullDCASPER takes on a similar maneuver as well.

5. Conclusions

The rapid growth of the human population and its changing distribution on Earth has placed more and more people in harm's way over time. Some of this harm can be traced to natural disasters such as earthquakes, floods and wildfire, some can be traced to technological failures such as nuclear accidents, and still other sources to the threat of war and other forms of human conflict. One possible strategy is to evacuate people and move them out of harm's way when one or more of these events threatens to unfold and more often than not, this usually entails moving people quickly with the help of various forms of motorized transport. The size and character of the disaster as well as the distribution of infrastructure and other facilities and people will have a large impact on the success of any evacuation and therefore the number of casualties. The size and capacity of the road network will loom large in both the planning and execution of an evacuation. In an earlier paper, the value of using the CASPER was demonstrated by modeling the ability of local road networks to move residents to safety when some form of disaster threatens.

We were enthused when we learnt that various public safety professionals had adopted and used the tool to evaluate how difficult it would

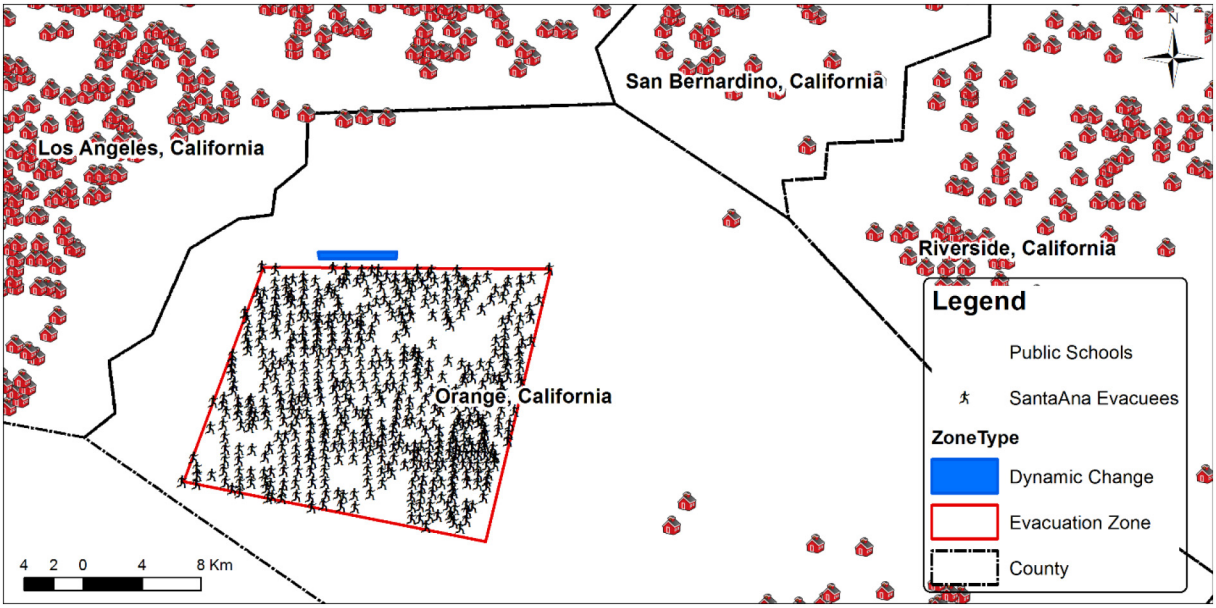

Fig. 5. Evacuation map for Santa Ana Downtown. All public schools that are at least 10 km away from fire are selected as shelters.

Table 2
All wildfire scenarios along with the selected populations, shelters, and dynamic changes.

Scenario name	Fire			Shelter count	Dynamic change			
	Area (km^2)	Neighborhood count	Vehicle count		Area (km^2)	Type	Effective time (min)	Road length (Km)
LADT[a]	243	794	285,978	4415	2.5	½ Lanes	15–300	30
SantaAnaDT	282	554	232,757	4737	2.1	½ Lanes	20–300	27
SDDT	359	546	201,536	4909	14.1	½ Lanes	20–300	62
All_DT	884	1894	720,271	4415	18.6	½ Lanes	15,20–300	119
SB	233	76	24,724	5284	2.0	Road Block	10–∞	19
SB_Ventura	608	105	37,837	5175	17.2	Road Block	10–∞	82
Riverside	2069	252	123,122	5066	23.1	Road Block	20–∞	164
LA	2364	571	210,254	4563	7.7	Road Block	10–∞	88
SD	3002	109	48,364	5028	30.1	Road Block	10,15–∞	305
LowerSoCal	5071	361	171,486	4761	53.2	Road Block	10–∞	470
Kern_SB	7939	93	28,971	5271	29.7	Road Block	10–∞	26
UpperSoCal	10,677	693	252,338	4392	52.6	Road Block	10–∞	177
All_Rural	15,748	1054	423,824	3820	105.7	Road Block	10,15,20–∞	647

[a] DT stands for downtown and refers to densely populated areas as opposed to rural areas.

Table 3
Running time and evacuation improvement for different SPT implementations ordered by average running time.

Setup	Average memory usage (MB)	Average EvcTime improvement (%)	Running time speed up (%)		Average running time (min)
			Mean	Std dev	
DsptCASPER_C02	1437.36	1.21	19.96	18.7	6.893
CDsptCASPER_C01	1784.80	1.55	**13.72**	15.8	6.901
CDsptCASPER_C02	1565.16	1.21	**20.29**	18.5	6.943
DsptCASPER_C01	1564.36	1.22	12.35	14.4	7.006
CASPER_C01	1649.00	1.22	7.78	11.1	7.349
CASPER_C02	1569.84	1.21	18.41	18.7	7.785

Table 4
Summery of running time and evacuation improvement between greedy and iterative CASPER.

Setup	Average memory usage (MB)	EvcTime improvement (%)			Average running time speed up (%)	Average running time (min)
		Mean	Std dev	Max		
CDsptCASPER	1686.54	1.99	4.56	21.65	44.69	8.136
DsptCASPER	1533.82	3.22	5.43	21.74	44.99	8.328
CASPER	1604.59	3.22	5.43	21.74	41.04	9.089
CItrCASPER	1697.76	8.12	16.26	95.96	20.72	11.919
ItrCASPER	1648.47	8.29	16.24	95.95	20.25	12.651

be to evacuate all of the residents threatened by a large disaster, such as a regional flood in the Sacramento-San Joaquin Delta for example. When used as a planning tool, CASPER can predict: (1) whether or not a disaster scenario is likely to cause traffic congestion that exceeds the capacity of the road network; (2) where these so-called "choke" points are likely to occur if the capacity of the network is exceeded; and (3) reasonable evacuation routes as long as the network is not saturated. The first outcome is fundamental and depending on the answer to this question, the second or third outcomes will provide public officials and emergency personnel with valuable information about the challenges that are likely to occur if such a disaster actually occurred.

The opportunities for coping with a saturated network (as in option (2) above) might include building new infrastructure to increase the capacity at predicted choke points or, in the case of certain kinds of events like regional floods that might unfold over a number of days, exploring ways to reduce traffic volumes, such as would happen if the evacuation could be accomplished in stages, so that these choke points are averted altogether. However, these outcomes were not considered for the work at hand and are therefore beyond the scope of this paper.

The work in this paper is focused on the cases in which the road infrastructure is such that we can plan an evacuation (i.e., as envisaged in option (3) above) and the shortcomings that were left unattended by the static solution. Indeed, the current work endeavored to solve the most substantial shortcoming with a static solution like CASPER, that it cannot capture and use knowledge of the disaster and/or the condition of the infrastructure as a disaster event unfolds. We should be able to take account of dynamic changes to the road network that might materialize when, for example, a bridge or overpass is taken out, in an earthquake or flood scenario and to evaluate the impact this likely to have on the efficacy of various evacuation scenarios.

We therefore have described the dynamic urban evacuation routing problem in this article and provided several possible solutions which were evaluated using a "worst case" wildfire disaster in southern California. The results of these experiments were conducted in two stages.

We first presented two classes of algorithms – the first using an iterative design and the second using a dynamic shortest path tree (DSPT) – to solve the static evacuation routing problem. DsptCASPER produced evacuation routes with similar quality faster than the original CASPER. Also, the ItrCASPER iterative design that produced routes that were about 8% better but needed more computation time compared to the fully dynamic DsptCASPER setup. We therefore recommend using iterative CASPER as it would produce better routes, but if running time were a priority, then DsptCASPER would give the fastest computation time.

We next turned our attention to the dynamic changes to the road network and compared the performance of four possible implementations of our new DCASPER solution. Among those, SmartDCASPER performed well in terms of both running and evacuation time. It will continue to

Table 5
Running time and evacuation improvement for all four different DCASPER implementations.

Setup	Average memory usage (MB)	Average EvcTime improvement (%)	Average EvcTime	Average running time speed up (%)	Average running time (min)	Average no. of stranded evacuee
SmartDCASPER	1871.89	**12.69**	130.07	**12.56**	6.921	7.21
DisDCASPER	1668.97	9.65	138.41	41.71	4.467	0.00
SimpleDCASPER	1568.55	7.01	147.03	45.82	4.165	0.79
FullDCASPER	1898.26	6.59	417.84	6.50	7.442	7.36

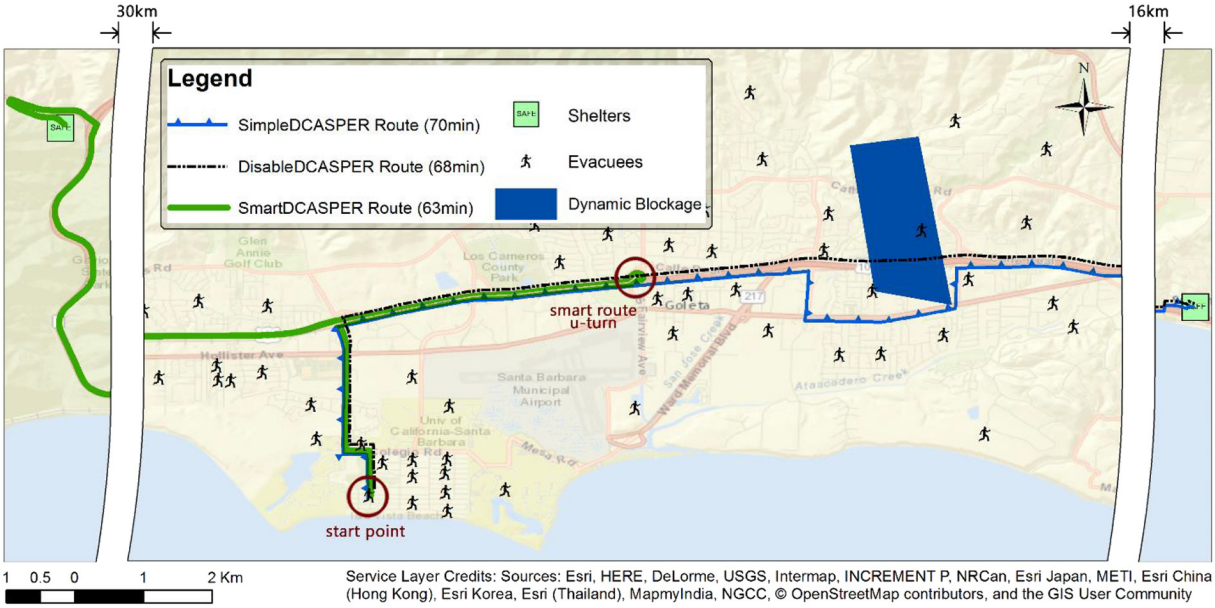

Fig. 6. Map of DCASPER evacuation of Santa Barbara, CA for one particular neighborhood. The SmartDCASPER route had to make a U-turn when the dynamic road blockage occurred. All three routes are drawn with a small offset from the road to avoid overlaps.

be a feasible solution so long as there are limited and infrequent changes to the network and the shelter locations remain unchanged.

Taken as a whole, the work shows that the proposed algorithms provide scalable solutions for the evacuation routing problem in a dynamic environment and that the results could be used to evaluate the impact of various infrastructure changes as part of the evacuation planning process that hopefully precedes disasters in high risk regions of the world.

Acknowledgments

This research has been funded in part by USC Spatial Sciences Institute (USC SSI). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the USC SSI. The authors would also like to thank the engineering team at Wood Rodgers Inc. for their helpful insights throughout the implementation of this research. For our experiments we used 2010 Census block group demographic data. The transportation network dataset was created from NAVTEQ NAVSTREETS Street Data 2007 with permission from The North American Association of Central Cancer Registries (NAACCR).

References

Bayram, V., Tansel, B.Ç., & Yaman, H. (2015). Compromising system and user interests in shelter location and evacuation planning. *Transportation Research Part B: Methodological*, *72*, 146–163. http://dx.doi.org/10.1016/j.trb.2014.11.010.

Bhaduri, B., Bright, E., Coleman, P., & Dobson, J. (2002). LandScan: Locating people is what matters. *Geoinformatics*, *5*, 34–37.

Chan, E. P. F., & Yang, Y. (2009). Shortest path tree computation in dynamic graphs. *IEEE Transactions on Computers*, *58*, 541–557. http://dx.doi.org/10.1109/TC.2008.198.

Chiu, Y. -C., Zheng, H., Villalobos, J., & Gautam, B. (2007). Modeling no-notice mass evacuation using a dynamic traffic flow optimization model. *IIE Transactions*, *39*, 83–94. http://dx.doi.org/10.1080/07408170600946473.

Church, R., & Cova, T. J. (2000). Mapping evacuation risk on transportation networks using a spatial optimization model. *Transportation Research Part C Emerging Technologies*, *8*, 321–336. http://dx.doi.org/10.1016/S0968-090X(00)00019-X.

Cova, T. J., Goodchild, M. F., Maguire, D. J., & David, W. R. (1999). GIS in emergency management. In P. A. Longley (Ed.), *Geographical information systems: Principles, techniques, applications, and management* (pp. 845–858). New York: John Wiley & Sons.

Esri (2015). ArcGIS for desktop [WWW document]. URL. http://www.esri.com/software/arcgis/arcgis-for-desktop, Accessed date: 17 June 2015.

Frigioni, D., Ioffreda, M., Nanni, U., & Pasquale, G. (1998). Experimental analysis of dynamic algorithms for the single. *ACM J. Exp. Algorithmics*, *3*. http://dx.doi.org/10.1145/297096.297147.

Frigioni, D., Marchetti-Spaccamela, A., & Nanni, U. (1998). Semidynamic algorithms for maintaining single-source shortest path trees. *Algorithmica*, *22*, 250–274. http://dx.doi.org/10.1007/PL00009224.

Frigioni, D., Marchetti-Spaccamela, A., & Nanni, U. (2000). Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, *34*, 251–281. http://dx.doi.org/10.1006/jagm.1999.1048.

Greenberg, H. (1959). An analysis of traffic flow. *Operations Research*, *7*, 79–85. http://dx.doi.org/10.1287/opre.7.1.79.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, *4*, 100–107. http://dx.doi.org/10.1109/TSSC.1968.300136.

Helbing, D. (2001). Traffic and related self-driven many-particle systems. *Reviews of Modern Physics*, *73*, 1067–1141. http://dx.doi.org/10.1103/RevModPhys.73.1067.

Holden, H., & Risebro, N. H. (1995). A mathematical model of traffic flow on a network of unidirectional roads. *SIAM Journal on Mathematical Analysis*, *26*, 999–1017. http://dx.doi.org/10.1137/S0036141093243289.

Kim, S., George, B., & Shekhar, S. (2007). *Evacuation route planning: Scalable heuristics* (pp. 1)Proceedings of the 15th annual ACM international symposium on advances in geographic information systems. New York, New York, USA: ACM Press. http://dx.doi.org/10.1145/1341012.1341039.

Kobayashi, T., Medina, R. M., & Cova, T. J. (2011). Visualizing diurnal population change in urban areas for emergency management. *The Professional Geographer*, *63*, 113–130.

Kwon, E., & Pitt, S. (2005). Evaluation of emergency evacuation strategies for downtown event traffic using a dynamic network model. *Journal of the Transportation Research Board*, *1922*, 149–155. http://dx.doi.org/10.3141/1922-19.

Leutzbach, W. (1988). Introduction to the theory of traffic flow. Berlin, Heidelberg: Springer Berlin Heidelberghttp://dx.doi.org/10.1007/978-3-642-61353-1.

Li, W., Zhu, J., Li, H., Wu, Q., & Zhang, L. (2015). A game theory based on Monte Carlo analysis for optimizing evacuation routing in complex scenes. *Mathematical Problems in Engineering*, *2015*, 1–11. http://dx.doi.org/10.1155/2015/292093.

Lu, Q., George, B., & Shekhar, S. (2005). Capacity constrained routing algorithms for evacuation planning: A summary of results. In C. Bauzer Medeiros, M. J. Egenhofer, & E. Bertino (Eds.), *Advances in spatial and temporal databases, lecture notes in computer science* (pp. 291–307). Berlin Heidelberg, Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/11535331_17.

Mahmassani, H. S., Sbayti, H., & Zhou, X. (2004). *Dynasmart-p version 1.0 user's guide. Maryl. Transp. Initiat. Coll. Park. Maryl.*

Malone, S. W., Miller, C. A., & Neill, D. B. (2001). Traffic flow models and the evacuation problem. *The UMAP Journal*, *22*, 271–290.

Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, *58*, 161–205 (10.1.1.56.6125).

Narváez, P., Siu, K. Y., & Tzeng, H. Y. (2001). New dynamic SPT algorithm based on a ball-and-string model. *IEEE/ACM Transactions on Networking*, *9*, 706–718. http://dx.doi.org/10.1109/90.974525.

Nassir, N., Hickman, M., Zheng, H., & Chiu, Y. (2014). Network flow solution method for optimal evacuation traffic routing and signal control with nonuniform threat. *Transportation Research Board*, *2459*, 54–62 (10.3141.2459-07).

Pel, A. J., Bliemer, M. C. J., & Hoogendoorn, S. P. (2011). A review on travel behaviour modelling in dynamic traffic simulation models for evacuations. *Transportation (Amst).*, *39*, 97–123. http://dx.doi.org/10.1007/s11116-011-9320-6.

Pourrahmani, E., Delavar, M. R., Pahlavani, P., & Mostafavi, M. A. (2015). Dynamic evacuation routing plan after an earthquake. *Natural Hazards Review*, *16*, 1–8. http://dx.doi.org/10.1061/(ASCE)NH.1527-6996.0000183.

Ramalingam, G., & Reps, T. (1996). An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*.

Santos, G., & Aguirre, B. E. (2004). *A critical review of emergency evacuation simulation models*Proceedings of building occupant movement during fire emergencies. Disaster Research Center.

Shahabi, K. (2015a). CASPER for ArcGIS [WWW Document]. ArcGIS Online. URL http://esri.com/arccasper, Accessed date: 17 June 2015.

Shahabi, K. (2015b). CASPER Source Code [WWW Document]. GitHub. URL http://github.com/spatial-computing/CASPER, Accessed date: 17 June 2015.

Shahabi, K., & Wilson, J. P. (2014). CASPER: Intelligent capacity-aware evacuation routing. *Computers, Environment and Urban Systems*, *46*, 12–24. http://dx.doi.org/10.1016/j.compenvurbsys.2014.03.004.

Smith, J. M. (1991). State-dependent queueing models in emergency evacuation networks. *Transportation Research Part B: Methodological*, *25*.

Smith, J. M., & Cruz, F. R. B. (2005). The buffer allocation problem for general finite buffer queueing networks. *IIE Transactions*, *37*, 343–365. http://dx.doi.org/10.1080/07408170590916986.

Stepanov, A., & Smith, J. M. (2009). Multi-objective evacuation routing in transportation networks. *European Journal of Operational Research*, *198*, 435–446. http://dx.doi.org/10.1016/j.ejor.2008.08.025.

U.S. Census Bureau (2010). Census population data [WWW document]. Census. URL. http://www.census.gov/popest/data/index.html, Accessed date: 17 June 2015.

U.S. Federal Highway Administration (2014). Delay-volume relations for travel forecasting: Based on the 1985 highway capacity manual [WWW document]. URL http://www.fhwa.dot.gov/planning/tmip/publications/other_reports/delay_volume_relations/ch04.cfm, Accessed date: 17 June 2015.

USC Spatial Sciences Institute (2014). US public schools GIS layer [WWW document]. URL. http://gis-server-01.usc.edu:6080/arcgis/rest/services/US_Schools/MapServer/1, Accessed date: 18 March 2017.

Xie, C., Lin, D. -Y., & Travis Waller, S. (2010). A dynamic evacuation network optimization problem with lane reversal and crossing elimination strategies. *Transportation Research Part E-Logistics & Transportation Review*, *46*, 295–316. http://dx.doi.org/10.1016/j.tre.2009.11.004.

Yang, K., Gunturi, V. M. V., & Shekhar, S. (2012). A dartboard network cut based approach to evacuation route planning: A summary of results. *Geographic Information Science*, *1*, 325–339.

Yi, W., & Ozdamar, L. (2007). A dynamic logistics coordination model for evacuation and support in disaster response activities. *European Journal of Operational Research*, *179*, 1177–1193. http://dx.doi.org/10.1016/j.ejor.2005.03.077.

Zhou, X., George, B., Kim, S., Wolff, J. M. R., Lu, Q., & Shekhar, S. (2010). Evacuation planning: A spatial network database approach. *IEEE Comput. Soc.*, *33*, 26.